Nexus Messenger

By Adonias Lopez & Henna Gohil

Intro

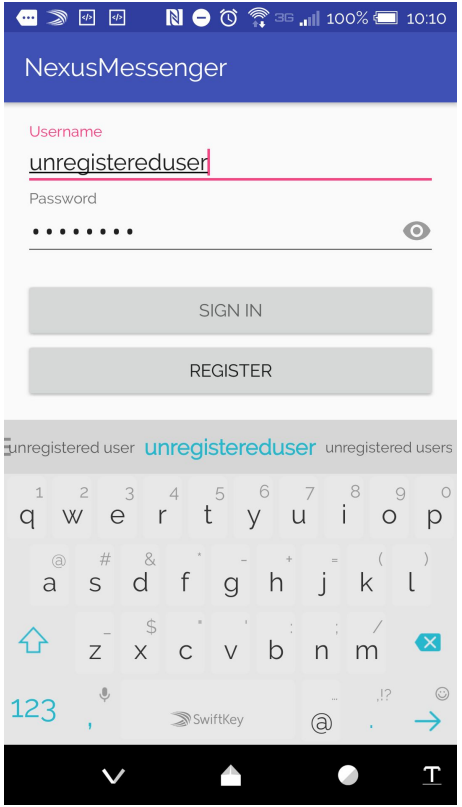
- We created an end-to-end secured android messaging app that allows a user to send a message to only one client at a time.
- This app allows a user to register by creating a unique name accompanied by a password, and then is taken to the login page.
- Once logged in, the user receives any pending messages that they had waiting.
- The user can then send a message to anyone he/she pleases through an exchange of keys through a QR code

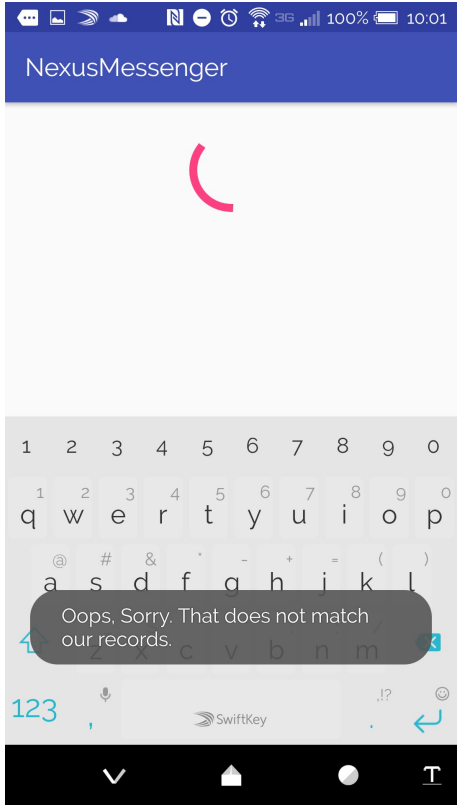
How to use Nexus Messenger 🔎

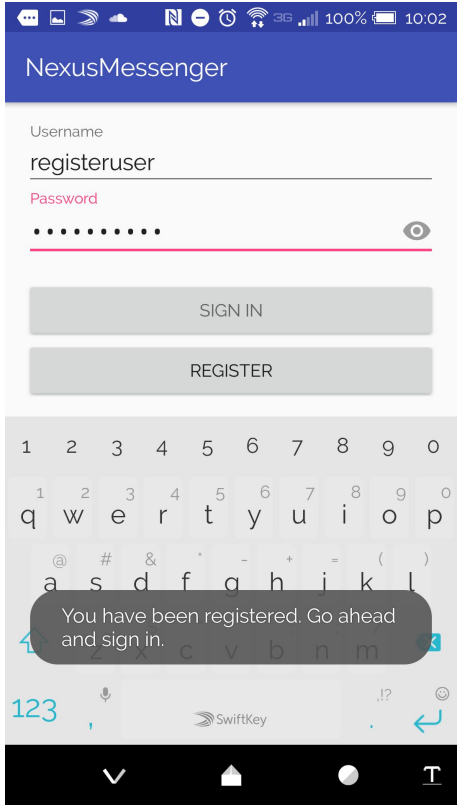
Getting Started:

1. Download/Install App
2. Once installed, you are brought to the Homepage and prompted to enter your login in the empty fields
3. You are given two options:
    a. Login
    b. Register (if user does not already have a login)
        i. Fill out main details
            1. Username (Unique)
            2. Password
4. Once Registered, you may log in

Login/Register Page



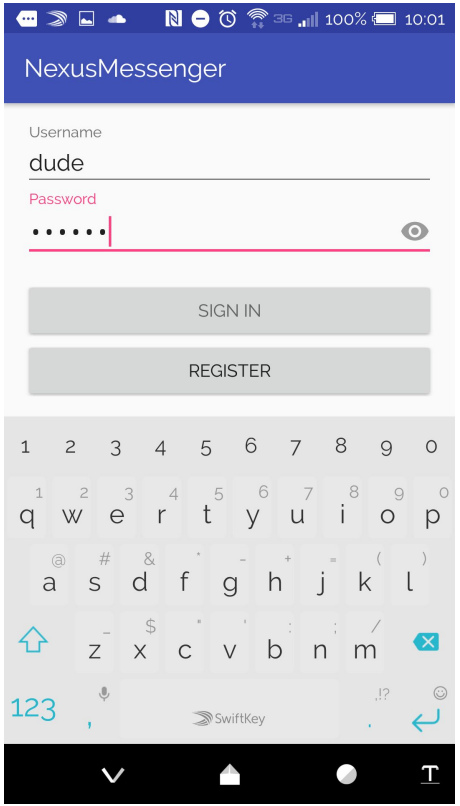


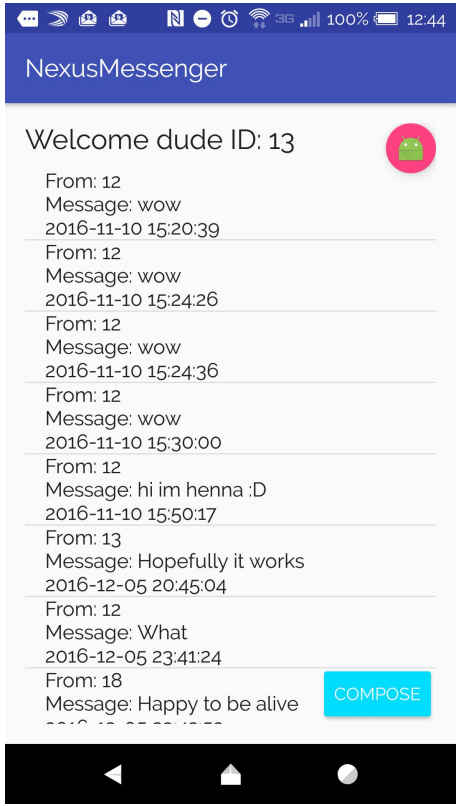


Login and Send Messages 📜

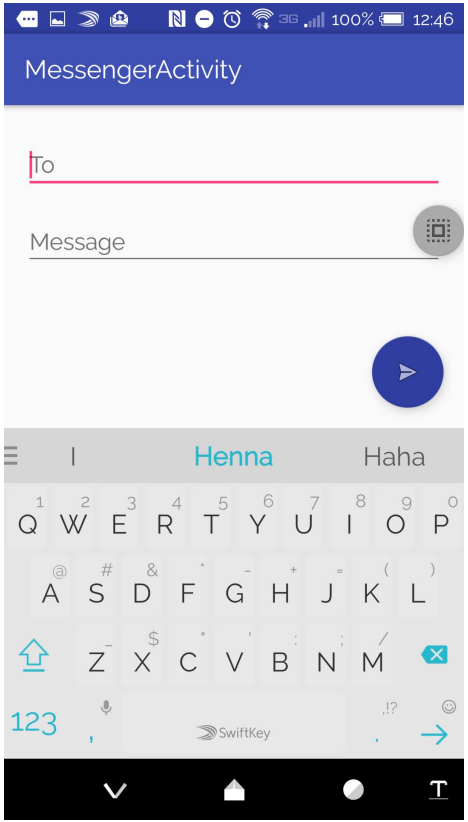
- Once logged in, the server will begin retrieving all the messages that were sent to the user.
- Then, the user will be given the option to send a message to a client.
- To do this, both users have to do a physical key exchange, QR code, where they take a picture of a unique code that is specific to that user.
- After the user is verified, and the content of message is written, user taps “Send” and the message is sent to that user.

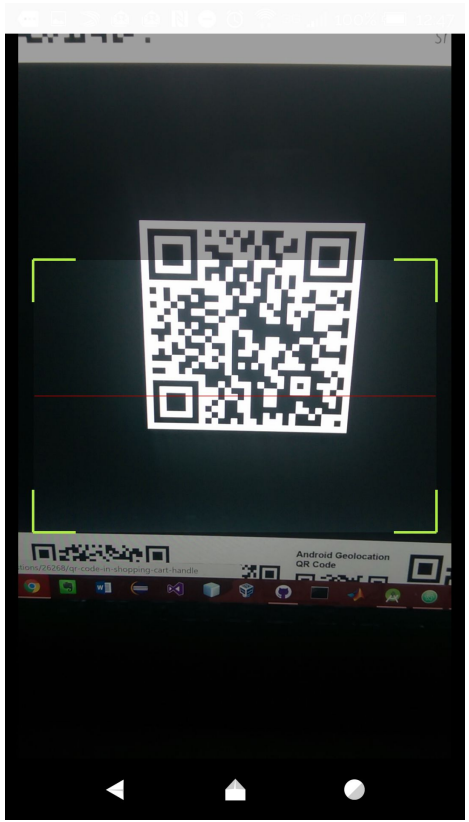
Logging in/Retrieving Messages

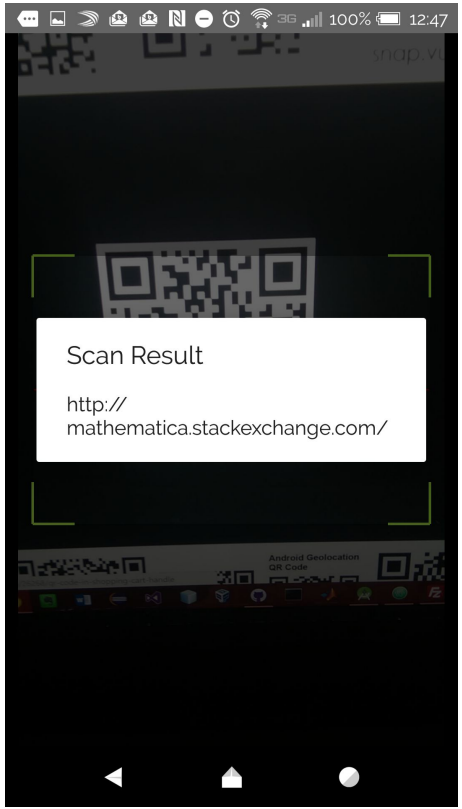




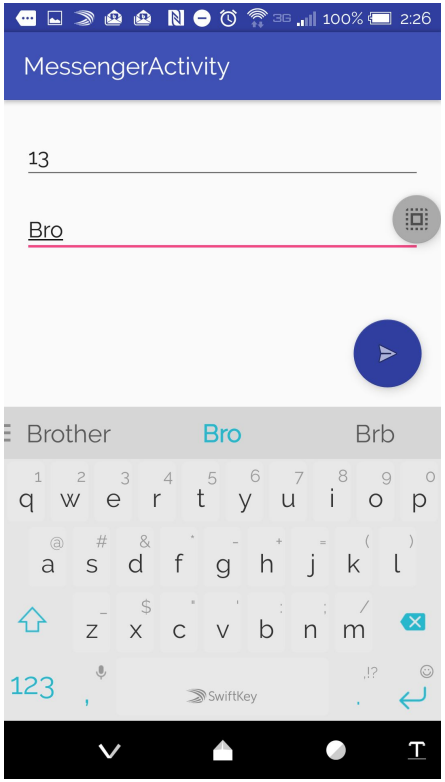
QR Code - Public Key Exchange

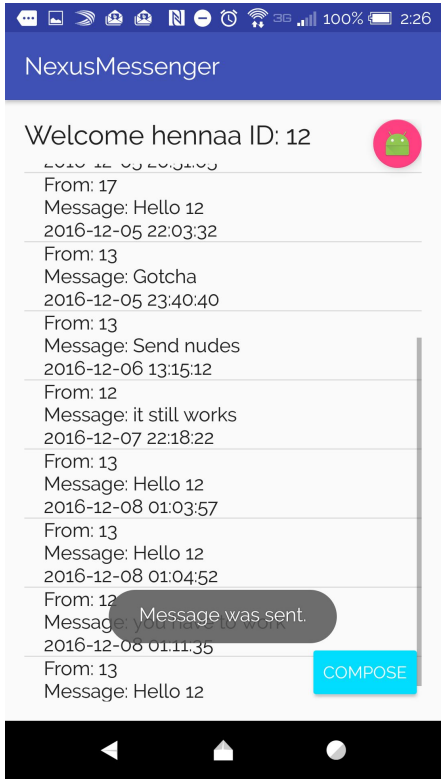


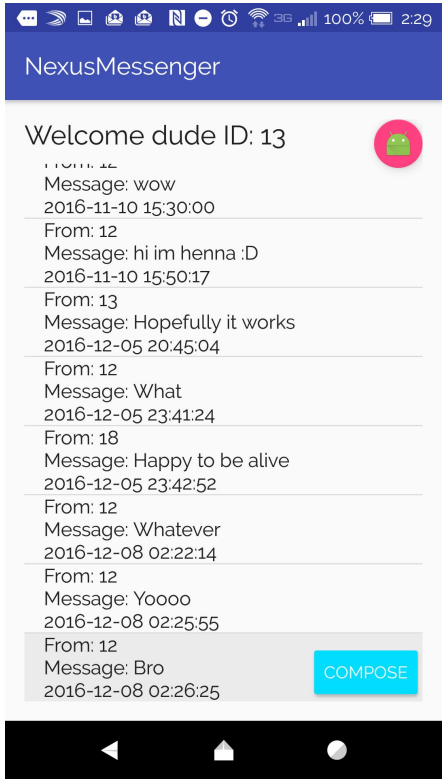




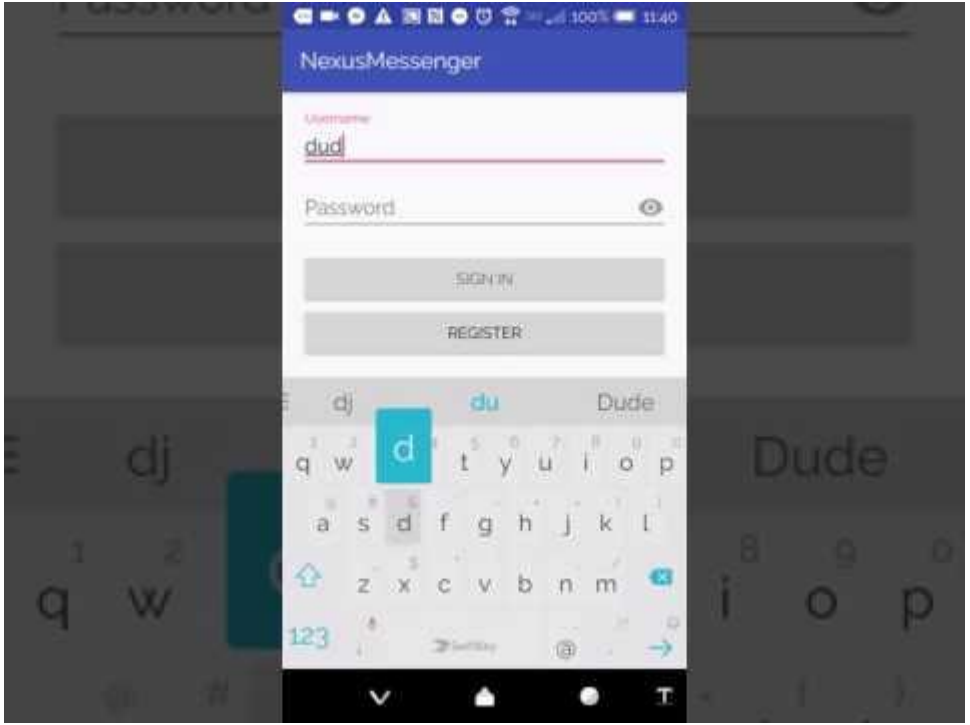
Message Sent! 







Video Demonstration on How it's Done



How does this work? □

There are three main components:

1. The Server
    a. Secured RESTful server to make sure all the data that is being passed is protected on the domain we created, nexusmessenger.pw
    b. Secured connection by implementing LetsEncrypt and TLS 1.2 by using certificates
2. The Database
    a. Used MySQL to store the user's
        i. Credentials
            1. Username, Password, UserID...etc
            2. Message Data
3. The Android App
    a. Utilizes both the server and the database to create a messenger app that sends and receives messages to a client

How it all began 🕐

Creating Nexusmessenger.pw

- Created our domain on Namecheap, and then hosted it on AWS by making an instance and connecting everything so that the local AWS site would redirect to our website
- Secured our site by using the AWS-provided CerbotOS SSH to install LetsEncrypt along with the TLS 1.2 through Linux commands
  - Edited ssl.config file to set secure configurations so that the Http server was always directed to Https (helped get the A+ for ssl labs)
  - Changed the ciphersuite to what the professor wanted, so that we didn't have things that were already broken on there.
    - Made our server extra secure with the strict transport security of Https

Cont'd

- OpenSSL protects the client-to-server communication and secures the integrity of the server
- LetsEncrypt created a Https server for our domain with a browser-trusted certificate that's automatically obtained, and from there we could easily revoke or renew certificates when they expired

Making the Server RESTful □ 💤 □

1. Used PHP to create GET & POST methods that would register a user and then log them in; used POSTMAN to test this.
   a. Created the User
   b. POST their login, GET JWT Token
   c. Passed bearer token as a workaround to not display the JWT token in the header
   d. To retrieve pending messages, the user GETs the messages waiting for them,
   e. To send a message, the user then POSTS messages when userID of receiver is already known
2. Making it stateless, due to the use of JWT tokens, nothing is stored on the server itself such as the session or cookies.

MySQL 💾

- Once user is registered using our nexusmessenger.pw domain, the user's credentials is saved and stored onto our MySQL database
- The data members that we store are:
  - UserID
  - ConversationID
  - MessageID
  - Password (Encrypted using bcrypt)
  - Username
- We've connected MySQL to our server so everything is updated once submitted.
- We've learned that encrypting the whole database is a bad idea :(

The Nexus Messenger App ✉ 💩

- Puts everything together, using the server and mySQL
- The App uses RESTful commands to connect to the server to login and retrieve messages.
- The app gets everything from the server but retains the public key within the device.
- The public key will be encoded into the QR and once it’s scanned it will be saved onto the device.
  - Used zxing library
- The sender’s public key will decode their messages depending on their username.
  - The user’s public key will be saved in a file on the device.
- QR exchange happens within the app, doesn’t use the server.

Cont'd

- Limited to Android 7.0-4.0
- We used SpongeyCastle for the public key exchange.

What Does our App Protect You From? 👹 🐍

- Outside Attacks
  - Uses end-to-end encryption to combat eavesdropping from outside attackers with LetsEncrypt.
- Inside Attacks
  - We have used endpoint authentication to combat Man-in-the-Middle (MitM) attack, such as the QR code for the public key exchange.

What we didn’t include from Phase 1 □

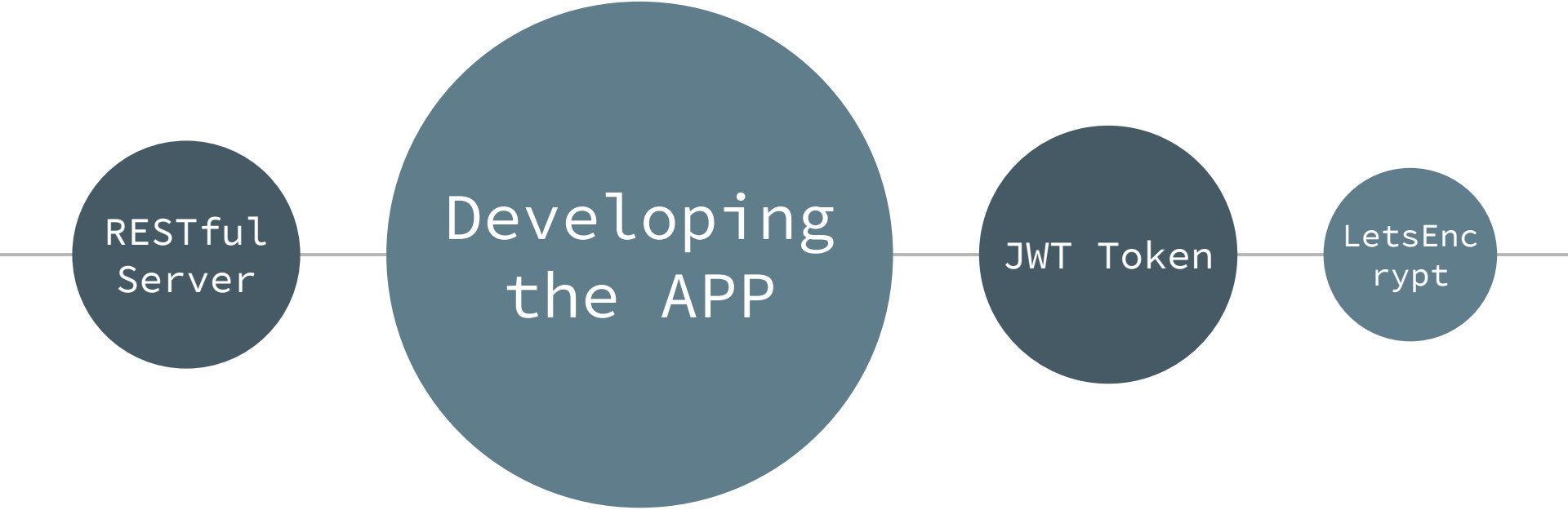
- Signal Protool from Open Whisper Systems
  - Thought we would use it to make the red tunnel of our system to secure the server, but didn’t end up using it
  - Why?
    - Looked too complicated and didn’t fit the necessary criteria for our app design
    - It was too advanced for the simple chat app we were creating
    - Same for WhatsApp since they use the same protocols

What we still have to Complete 💻

OpenPGP (Pretty Good Privacy)

- Our green tunnel
- We've implemented the QR code into our app but haven't encrypted our messages yet.

Level of Difficulties on this Project



Any Questions?